RECOGNIZING ELEMENTARY ELEMENTS IN CHEMICAL DIAGRAM SKETCHES

An Undergraduate Research Scholars Thesis

by

ELLIE MILLER

Submitted to the Undergraduate Research Scholars program at
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by Research Advisor: Dr. Tracy Hammond

May 2018

Major: Computer Science and Engineering

TABLE OF CONTENTS

ABSTRACT

Recognizing Elementary Elements in Chemical Diagram Sketches

Ellie Miller
Department of Computer Science and Engineering
Texas A&M University

Research Advisor: Dr. Tracy Hammond
Department of Computer Science and Engineering
Texas A&M University

Organic Chemistry is a challenging subject that requires dedicated practice to learn the meticulous rules composing the subject, otherwise a student risks failure. Current software to teach chemical structures contains drag-and-drop components and fails to provide students with true understanding of Organic Chemistry concepts. My solution is to integrate a sketch recognition interface that can learn to recognize components of various, user-sketched chemical structures with a back-propagation neural network that can be trained to translate the components of the chemical structure sketch to determine correctness. The accuracy of the program will be rigorously tested to determine correctness in interpreting chemical structures.

DEDICATION

To my mom and dad, who have always supported me.

ACKNOWLEDGMENTS

I would like to express my unending thanks to Paul Taele for his constant support and encouragement throughout this project. Paul has provided thoughtful direction and constructive suggestions since I began my work. I would also like to thank my research advisor, Dr. Tracy Hammond, for her professional guidance and advice. I would like to thank Dr. James Pennington for advising me on what components of Organic Chemistry his students struggle with and for clarifying topics of Organic Chemistry I was rusty on.

NOMENCLATURE

ITS	Intelligent Tutoring Systems
GUI	Graphical User Interface
ANN	Artificial Neural Network

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

Organic Chemistry is a notoriously difficult subject that requires dedication and meticulous practice to learn the rules of the subject [1, 2, 3]. The current software available to students to introduce them to these rules fails to allow students the freedom to hand draw chemical diagrams and correct errors. There are generally two types of software available to students for them to practice Organic Chemistry structures: drag-and-drop and hand drawn sketch recognition [4, 5]. Drag-and-drop software is most common for professors to assign homework through, but their features restrict the user. While drag-and-drop software is capable of identifying if a structure is correct or not, it is unable to specifically identify where in the structure an error occurs. Hand-drawn sketch recognition software generally provides users the freedom in physically drawing the structure, but has no correction evaluation component. Since current drag-and-drop software works well enough for professors to use for homework assignments, there can be little motivation to advance the software to incorporate more intelligent sketch recognizing. Hand-drawn sketch recognition software for chemical structures can be expanded to include correction evaluation for drag-and-drop software and provide concise, multi-functional software for students to better learn Organic Chemistry.

The complexities of Organic Chemistry are built around the basics of correctly drawing and identifying organic chemical structures. Organic Chemistry compound structures include elements from the periodic table of elements, lines representing shared electrons, and rings in the shape of triangles, pentagons, hexagons, and so on for structures with a ring formation [6]. Shorthand line notation can be used to omit the symbols for carbon and hydrogen and simplify the structure. Each structure must be balanced according to its respective chemical formula and must be oriented according to its molecular name. Neural

networks have been used quite successfully for modeling molecules and predicting organic chemistry reactions [7, 8, 9]. Neural networks have a long history of use in successful alphabet recognition [10, 11, 12]. In this application, neural networks will be pivotal for correct identification of chemical structure components.

In this thesis, I propose a solution for use in improving software for Organic Chemistry students. The solution involves integrating sketch recognition software with techniques for identifying the correctness of chemical structures. This also involves several different components to correctly recognize and evaluate a user drawn organic chemistry sketch. The remaining contents of the thesis will detail specifics of related works similar to this work, the design and algorithms of the proposed techniques, the results of evaluating the proposed techniques, and discussions regarding follow-up work for advancing the work of this thesis.

2. RELATED WORK

Organic Chemistry involves detailed drawings of organic compounds for use in chemical reactions. In teaching the subject of Organic Chemistry, professors and teachers must analyze these diagrams to determine correctness and student understanding. Since a single chemical diagram can be drawn several ways and still be a correct interpretation, analyzing these diagrams is a tedious and time-consuming task. The same holds true for other courses that involves material that can be interpreted in several different ways and still be correct (i.e. free body diagrams, East Asian languages). Therefore, many professors use online systems called Intelligent Tutoring Systems (ITS) for homework and quizzes to simplify their teaching load.

2.1 Chemistry Intelligent Tutoring Systems

Current Organic Chemistry ITS involve drag-and-drop systems that simplify the interpretation required for the software to determine correctness. The user is provided an interface with various tools and options to create their chemical structure. The user first drags and then drops various components to generate their desired chemical structure, and the interface is subsequently able to determine the structure correctness. There exists diverse ITS development that implements this technique including ChemDoodle and ChemSketch (Figures 2.1 and 2.2).

The ChemDoodle interface involves selecting which component that the user wants to draw, then clicking on the canvas where the component is desired. The ChemSketch interface offers similar functionality with pre-built chemical compounds and file-saving capabilities. The ChemDoodle interface is used with systems such as OWL that allows for correctness evaluation. However, error detection only denotes when a compound is incorrect and does not provide details on where the error lies. The ChemSketch interface is only

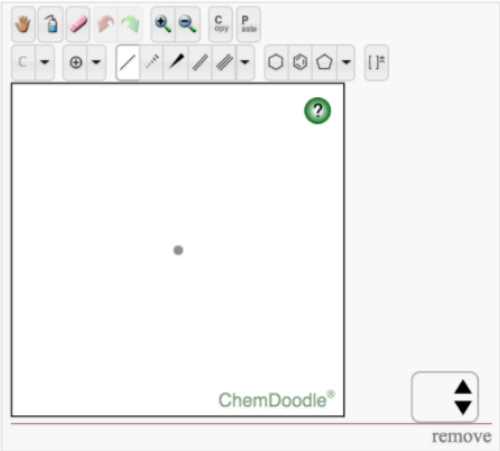

Figure 2.1: Screenshot of ChemDoodle interface.

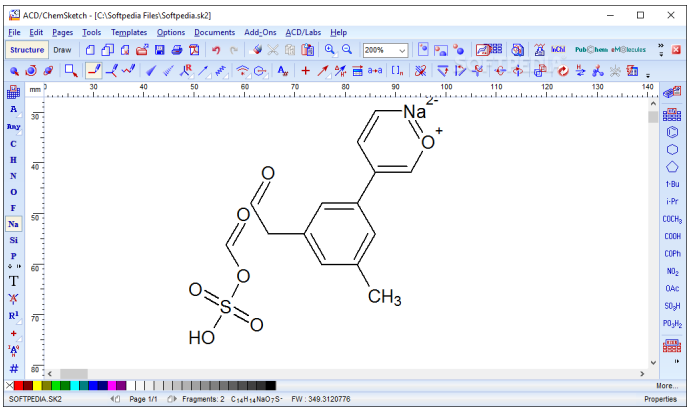

Figure 2.2: Screenshot of ChemSketch interface.

designed for generating chemical structures and does not provide correctness evaluation directly.

2.2 Chemistry Diagram Recognition

Recognition software can interpret hand-drawn chemical diagrams and interpret the components in an ideal form. One such software by Ouyang et al. [5] has a trainable symbol recognizer that analyzes the spatial context of each symbol, and THEN uses knowledge

of the domain to predict the correct components of the structure. Each individual stroke is examined with up to seven other strokes to determine which stroke combination makes up a valid component. Text recognition is completed by a combination of a commercial Microsoft recognizer and a distance metric similar to the Tanimoto coefficient. The strokes determined to not be alphanumerics are recognized by a recognizer trained on valid symbols such as atomic element lines, straight bonds, hash bonds, and wedge bonds. Once each component is parsed and recognized, the system then uses basic chemistry knowledge to determine if the structure is valid. For example, N_2N is an invalid chemical structure. The system will test possible scenarios to determine if it misinterpreted a component, but will ultimately allow for these incomplete structures since the user might choose to leave the structure inconsistent when they know what the remainder is.

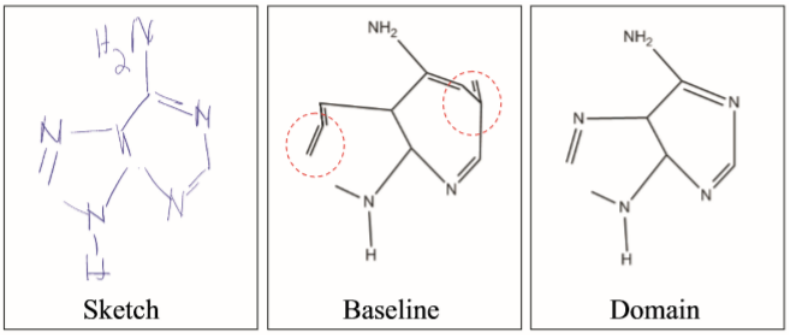

Figure 2.3: Process of recognizing hand-drawn chemical structures.

Figure 2.3 displays the process of chemical structure recognition. The sketch, in the first panel, is converted into a baseline structure, in the second panel, and any inconsistencies with domain knowledge are corrected, as seen in the third panel. This software allows for user freedom in generating the chemical structure, but it does not allow for error detection. It assumes inconsistencies are intentional, and does not output an error

message denoting how the structure is violating domain rules. However, the system does allow for stereochemistry and is able to recognize hash bonds and wedge bonds, which are particularly difficult to parse.

2.3 Sketch-Based Intelligent Tutoring Systems for Other Domains

Sketch-based ITS exist beyond the domain of chemistry. There is a diverse spectrum of successful sketch-based ITS for educational applications [13, 14] that encompass many other domains such as Mechanical Engineering [15, 16, 17, 18], children sketching [19, 20, 21, 22, 23], East Asian writing [24, 25, 26, 27, 28, 29, 30], engineering design [31, 32, 33, 34], music [35, 36], math [37, 38, 39], and art drawing [40, 41].

Mechanix [16] is a sketch-based ITS for drawing free-body diagrams, an application very useful for engineers in statics courses. This software allows professors to draw a correct answer, ensuring that students receive feedback for their planar truss and free body diagrams. Students can correct their work based on the feedback and resubmit until the diagram is correct, proving all objectives have been learned. The system is even implemented in exams, and the error-feedback system allows students to learn from mistakes rather than memorizing information.

PerSketchTivity [31] is a similar software designed for engineers to develop design sketching skills. This ITS first teaches users how to draw lines, circles, and squares before teaching 2-point perspective and then 3D visualization of complex shapes. During these lessons, PerSketchTivity monitors and analyzes the user's progress to ensure students perform physical movements with consistency. The software provides immediate feedback as students progress towards the solution. Students using this software generally had a positive opinion towards sketching and indicated that their confidence in sketching, creativity, and visualization had improved.

Sketch-based ITS also exist for learning symbol writing, including Japanese (e.g.,

Hasigo [24]) and music theory (e.g., Maestoso [35]). Hashigo works to prevent bad learning habits for students mastering the visual structure and written technique of Japanese kanji. This software provides critique and feedback on the structure itself as well as the student's written technique in sketching kanji. The feedback allows students to identify specifically where their errors lie and prevents bad practices from being developed. Maestoso teaches music theory through sketching practice. The sketch recognition software of Maestoso recognizes music structure elements and provides feedback for students learning these introductory music theory concepts.

The most significant aspect of all these aforementioned sketch-based ITS is the error feedback for the user. This feedback is how the user learns correct technique and domain rules to be successful in the respective domain. Without sufficient error recognition and feedback, a user will continue making the same errors without learning from their mistakes. The benefit of sketch-based ITS lies in the ability for users to learn the rules of the domain through sketch-based practice.

3. DESIGN

The focus of this chapter involves my exploratory efforts of the problem statement related to ITS for Organic Chemistry. I first investigate the domain of Organic Chemistry by consulting with an expert and students in the subject. I later investigated how users may perform the sketching necessary in the proposed solution through a paper sketching study, so that its insights can better form the design of the solution.

3.1 Interviews and Observations

For the interview, I reached out to an expert in the subject of Organic Chemistry, Instructional Associate Professor Dr. James Pennington of Texas A&M University's Department of Chemistry. I interviewed Dr. James Pennington as I began narrowing the scope of my domain. I consulted him to discover what topics of Organic Chemistry that most students struggled with most. Dr. Pennington advised me of the benefits of creating a good foundation of Organic Chemistry, starting with the basis of chemical structures. He also stated that stereochemistry is a challenging topic to most students. The SMILES enumeration does work for stereochemistry, but the challenge for allowing stereochemistry would lie in interpreting the hash and wedge bonds. Therefore, stereochemistry is outside the scope of this project, but it is a potential area for exploration as a future work direction of the thesis. This potential for adding stereochemistry is further discussed in Chapter 7.2: Stereochemistry.

Dr. Pennington specifically acknowledged that students struggle with organic reactions. I feel my application can be used to help teach students chemical reactions if the reaction structures and product structures are known. The system can then check if the reaction and product structures were drawn correctly. However, with the current state of the software, each reactant would have to be drawn separately and the product also drawn

separately. Each sketch is interpreted as a single chemical structure. The software could be extended to accept multiple structures drawn in one interface, but this would be added in future work.

In my interview with Dr. Pennington, he also discussed that the error recognition of the current homework software is less than ideal. The software will output when a student's structure is incorrect, but fails to give detailed specifics about the error of the structures. Therefore, students can often make similar errors several times before the system counts the attempts as a failure and provides the student with the correct answer. Students need more detailed error messages, such as where in the structure the error is located and what the specific error is (misplaced element, missing element, etc.), to improve their learning. Therefore, my software is designed with the intention of having very clear feedback to the user when a mistake is made. The error will be graphically displaced with a text output describing the error.

I spoke with several current Chemistry and Organic Chemistry students. One of the main themes that we discussed was how time-consuming drag-and-drop ITS were. The drag-and-drop systems require students to hand-draw the chemical structure on paper first before using the system to drag and drop the correct components on the system. Students find this very tedious, especially if it takes several attempts to draw the correct structure. When discussing the error recognition with these students, they appreciated the design concept of specific error messages for incorrect structures. The consulted students felt this would improve their understanding of the topic.

3.2 Paper Sketching Study

I did a very abbreviated study to analyze how individuals draw hexagons and other components of an Organic Chemistry compound. After my abbreviated study, I found that on a blank canvas, only 6/10 people drew a hexagon structure in one stroke. However,

with a honey-comb background grid, 9/10 people drew a hexagon structure in a single stroke. Regardless of the background of the canvas, most of the letters were drawn with intersecting lines. This led me to design the sketch interface with a large beehive background to guide the user so they will draw any aromatic, hexagon rings in a single stroke. The honeycomb background is shown in Figure 3.1. A hexagon in a single stroke is much easier and faster to interpret than a hexagon drawn in several individual strokes. Sketch recognition becomes much more complex with multiple strokes per shape because the recognizer has to identify which strokes should be interpreted as a group rather than interpreted as a single stroke.

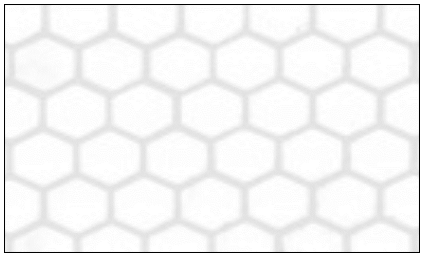

Figure 3.1: Honeycomb background of the GUI.

There are four types of drawn hexagons, all shown in Figure 3.2.

The single stroke for hexagon 3.2A is ideal for the sketch recognizer. The other hexagon types are less ideal for the recognizer. The multi-stroke hexagon 3.2B would require the recognizer to first identify which strokes should be evaluated together as a hexagon before the hexagon tests are administered. These strokes may or may not overlap, and because of this variance, a multi-stroke hexagon is extremely difficult to recognize. The multi-stroke hexagon with an additional line,Figure 3.2D, is similarly difficult for a recognizer to analyze since some of the strokes combine to make a hexagon, but there

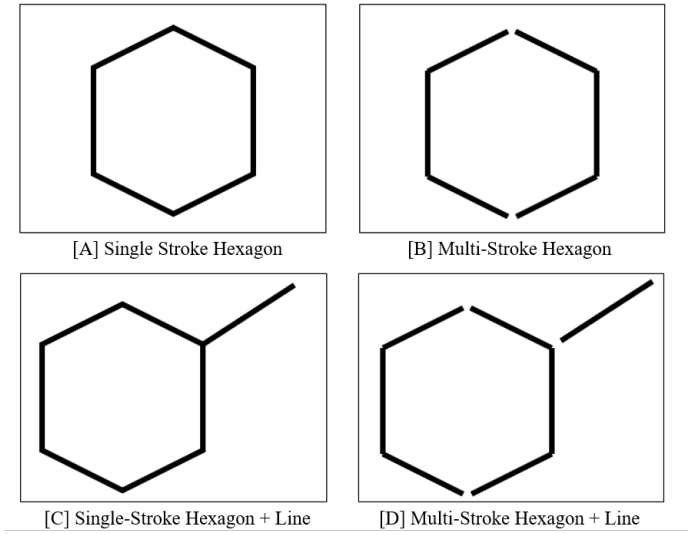

Figure 3.2: Four types of user-drawn hexagons.

is an additional line that should be interpreted as a shared electron line. The single-stroke hexagon with an additional line, Figure 3.2C, is a hexagon and shared electron line drawn in one stroke. This is difficult because the single stroke has to be broken into its individual components in order to be parsed correctly. Correct identification of the individual components of a chemical structure is the key to accurately recognizing if a chemical compound is drawn correctly.

I chose to focus on the single stroke hexagon in Figure 3.2A due to time constraints and the other major components of my software. The other types of hexagons can be examined as a future works project, but remain outside the scope of this project. This project will focus on ideally drawn shapes, such as single stroke hexagons, single stroke pentagons, etc. Single stroke shapes will be easily identifiable for the recognizer as opposed to multi-stroke shapes. With shapes assumed to be single strokes, the only multi-stroke objects my recognizer will have to identify are letters.

4. APPROACH

4.1 Sketch Data Structure Hierarchy

A sketch is composed of a hierarchy of objects (Figure 4.1).

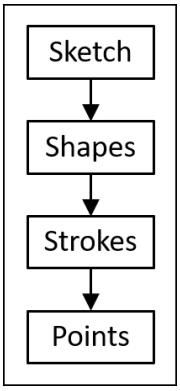

Figure 4.1: Hierarchy of the sketch.

This sketch hierarchy allows for top-down recognition of the sketch [42]. The base foundation of a sketch is the points collected as (x, y) coordinates. These points compose a stroke and multiple strokes are interpreted as a shape. Multiple shapes make up a sketch. The recognizer must first take the points collected from the Graphical User Interface and resample these points to gather enough points for recognition. The resampling methods are discussed further in Section 4.4: Resampling Algorithms. The sketch recognizer must determine which strokes compose a shape and then test this shape accordingly. If the shape is considered to be an alphanumeric, then the shape is sent to the trained neural network for appropriate recognition. If the shape is considered to be a geometric shape (i.e., a hexagon or line), then the shape is tested through various geometric shape algorithms for correct identification.

4.2 Backpropagation Neural Network Algorithm

The system uses a simple feedforward back-propagation neural network for identification of alphanumeric symbols (Figure 4.2).

```
Function TRAINNETWORK()
      repeat
            for SampleNumber = 0 to NumberOfSamples
                  for all nodes in Layer[0].Node.length
                        Layer[0].Input[node] ← Input[SampleNumber][node]
                  end for
                  FEEDFORWARD()
                  for all nodes in Layer[NumberOfLayers-1].Node.length
                        ActualOutput[SampleNumber][node] ←
                              Layer[NumberOfLayers-1].Node[node].Output
                  end for
                  UPDATEWEIGHTS()
                  //excluding details about killing thread function
            end for
            CurrentIteration ← CurrentIteration + 1
            CALCULATEOVERALLERROR()
      until ((OverallError > ThresholdError) &&
              (CurrentIteration < MaxNumberIterations))
end function
```

Figure 4.2: Neural network training function.

The training function of ANNs is pivotal to generating relationships between given input and expected output [43]. This function fills the first input layer of neurons with data from the given sample. After the input data has been stored, the network performs the feedfoward operation using the output of each layer as the input to the next. The feedfoward routine uses the weights and bias in the sigmoid function to calculate the appropriate output for the node. This output is then passed as the input to the next layer of nodes. The training then stores the output of the feedforward function to its corresponding output vector. The routine updates the weights of each node to calculate the signal error

for the output and all nodes in the hidden layer and also calculates the error from the calculated weights and bias against a predetermined threshold. This function trains the network on the input data so the neural network will be capable of anticipating outputs based on the relationships the network created during training.

4.3 Graphical User Interface

After my small paper sketching study, I chose a hexagon honeycomb background as an assistive visual cue for the Graphical User Interface, so users have a hexagon shapes as visual reference for easy tracing of aromatic rings. The GUI also contains buttons that were used throughout testing and for the user's availability. These buttons have corresponding functions such as drawing bounding boxes, finding overlapping strokes, and resampling strokes.

4.4 Resampling Algorithms

A stroke is a collection of (x, y) points. Each stroke collected from the GUI gathers these (x, y) points at various time intervals, meaning that sometimes a stroke is missing points. Figure 4.3 shows an example of this.

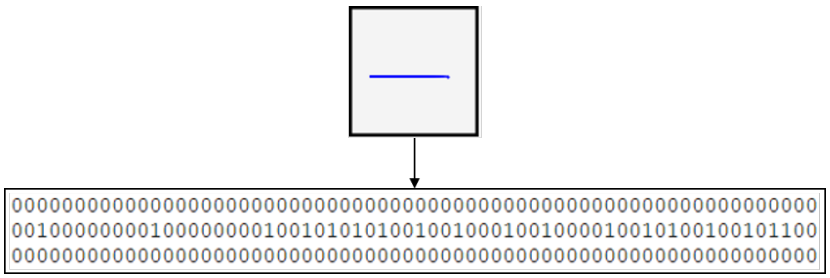

Figure 4.3: Line drawing example.

The line drawn on the GUI is straight and was drawn in one stroke, meaning there

are no gaps in the line. However, the (x, y) points that were grabbed by the software shows gaps due to the slight delay in events between what the user has drawn and what the system has detected. Therefore, empty space, represented by 0's, breaks the user stroke, represented by 1's, even when the stroke appears complete. Therefore, I generated resampling methods to reduce the number of unintentional gaps in strokes. Resmpling allows for strokes drawn at different speeds and varying levels of connectedness to be comparable [44].

The first resampling method checks the distances between each of the collected points for a particular stroke. A collected point is compared with its neighbor and the distance between the two is calculated. If the distance between the two is greater than five pixels, then an additional point is added halfway between these points. This reduces the accidental spaces in each sketch, as each stroke has additional points added as needed.

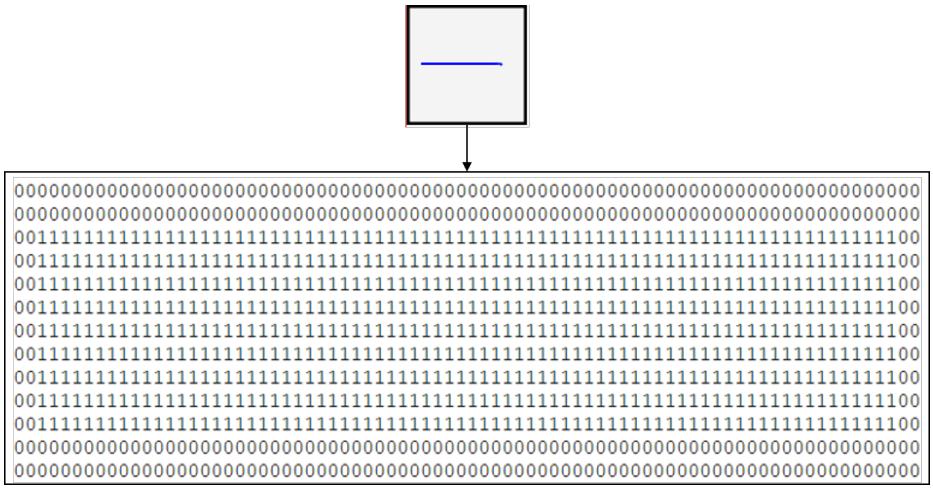

Figure 4.4: Line resampling.

The second resampling method adds extra points around each collected point. After additional points are added to fill in any gaps, each single (x, y) point of a sketch has a

nine-by-nine block of 1's pixels added around it. This also fills in any remaining gaps that the distance resampling failed to fill in.

Figure 4.4 displays what the sketch looks like to the system after both resampling methods are applied to the user's sketch. The increase in points per sketch is especially helpful when the neural network is working to identify an element of a sketch.

4.5 Bounding Boxes

The system has to identify which strokes should be evaluated together as a single shape. Using bounding boxes helps with this identification. A bounding box evaluates an entire stroke and encompasses it in a surrounding box. An example of the system's bounding boxes function can be seen in Figure 4.5.

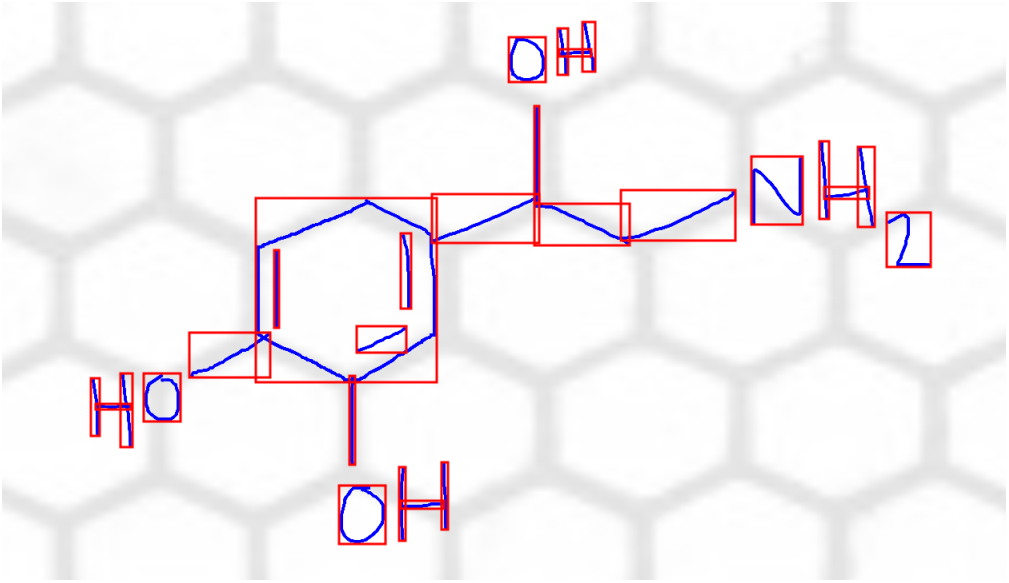

Figure 4.5: Bounding box function example.

Bounding boxes provide important information to the system about stroke relationships [42]. These bounding boxes are used in determining which strokes to test as a

collection of strokes comprising a shape. If the bounding boxes are intersecting, they are assumed to be a shape together and tested as such. This algorithm is under the assumption that letters that are drawn with multiple strokes have all the strokes intersecting. This will have bounding boxes that are intersecting, leading to the strokes being tested as a shape. If the shape with multiple strokes is tested and no valid recognized option is given, different strokes are tested together as a shape.

4.6 Hexagon Detection Algorithm

For each desired shaped of the system, a set of conditions must be determined for testing of that shape [45]. There are several conditions a hexagon must satisfy to be determined as a hexagon. The first condition is a square bounding box. An ideal hexagon has a bounding box that is approximately square. Therefore, the first check condition is to determine the bounding box of the shape and see if that box has a ratio of width to length that is roughly equal to 1 ± 0.1 (to account for slight discrepancies in the drawing). Figure 4.6 displays the square bounding box that accompanies a hexagon.

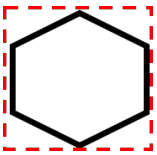

Figure 4.6: Bounding box of a hexagon.

Another condition that a hexagon must satisfy is that a hexagon has six corners. Therefore, a stroke is tested to see if it has 5-7 corners (the range is to account for possible noise in the stroke). The ShortStraw algorithm was implemented in the system to find corners [46]. Points begin to tighten around corners as the user slows the stroke to draw the corner. Therefore, finding segments of the stroke where the distance between points is

small relative to the overall stroke, corners can be determined. This method determines the corners of a potential hexagon and allows for this second condition to be tested. Therefore, if the bounding box of the hexagon is not ideal and square, the hexagon can still be detected by the algorithm.

4.7 Determining Strokes Comprising Shapes

A large component of the sketch recognition software is the success of correctly identifying each stroke as a substroke of a shape class [45].

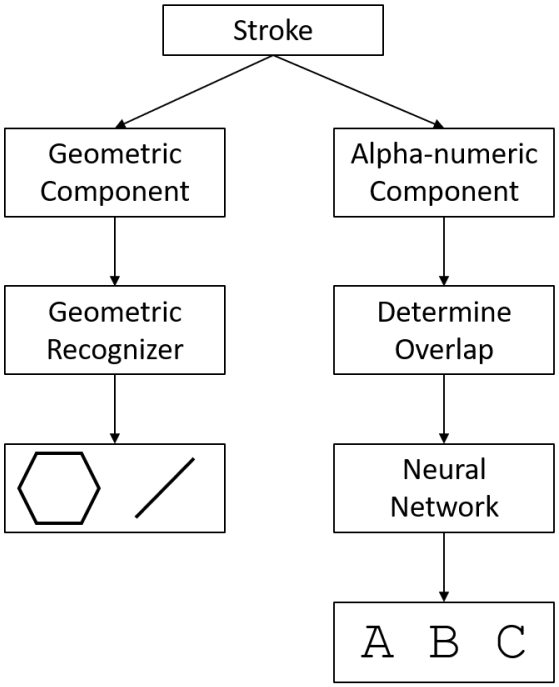

Figure 4.7: Process of recognizing strokes.

The basic hierarchy of characterizing a stroke in this system can be seen in Figure 4.7. The recognition begins by identifying each individual stroke and determining which recognition algorithms must be applied. I am using the assumption that geometric shapes are

drawn with a single stroke. This simplifies the logic used for determining if a single stroke should be identified as a geometric shape or as a component of an alphanumeric. Each stroke of a geometric shape is typically larger than that of an alphanumeric stroke. This simple logic case allows the system to send the stroke to the geometric recognizer or to the alphanumeric recognizer. A stroke could also be passed through the geometric recognizer first to see if any of the algorithms identify the stroke as a geometric shape. If the stroke passes through the geometric recognizer algorithms and none of them are able to identify the stroke, then the stroke must be an alphanumeric component. The alphanumeric recognizer must find the overlapping additional stroke components. Many letters have multiple strokes and must be sent to the neural network as a collection of strokes in order for correct identification. If the stroke is determined to be part of an alphanumeric, an algorithm determines if there are any overlapping components. The system assumes that all letters with multiple strokes are drawn with overlapping components.

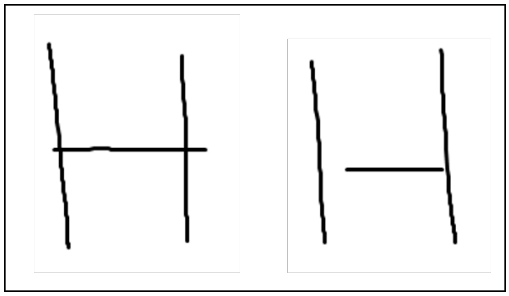

Figure 4.8: Overlapping strokes of the letter 'H'.

For example, Figure 4.8 displays two differently drawn H's. The one on the left has all the strokes overlapping with one another. The one on the right has strokes that do not overlap with one another. For simplicity of the alphanumeric recognizer, it is assumed that all letters composed of multiple strokes are all overlapping one another. This way,

the algorithm can simply identify which bounding boxes overlap one another to determine which strokes comprise an alphanumeric. Once the strokes of an alphanumeric are grouped together, they pass through the neural network for identification.

5. RESULTS

The focus of this chapter is on evaluating the different aspects of my proposed techniques in the thesis approach. Specifically, I discuss my data collection and the evaluation of my recognition algorithm's performance.

5.1 Collected Samples for Testing

I created a small 100 pixel-by-100 pixel GUI so that users could draw each letter—capital and lowercase—and the numbers 2 and 3. The neural network was trained with 15 samples for each of the stated alphanumeric classes that were drawn visually similar to the Courier New typeface. Each letter initially had 10,000 points, which were used as a testable file with 10,000 features. I then created an algorithm to count the frequency of 1's in each 100-by-100 pixel block of the letter. This translates the 10,000 pixel letter into a 10-by-10 block of frequencies.

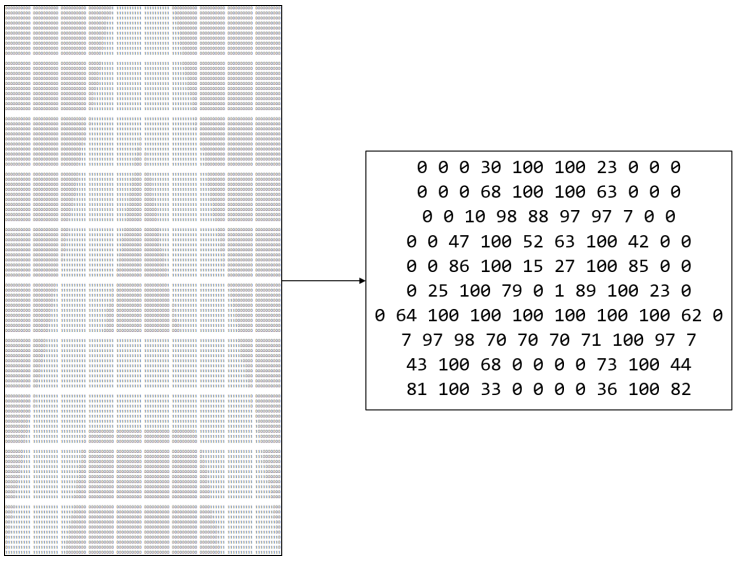

Figure 5.1: Letter translated to a 10-by-10 frequency block.

Figure 5.1 displays a letter transformed into its corresponding 10-by-10 frequency block. The 822 collected hand drawn alphabet samples were then translated into their corresponding 10-by-10 frequency blocks for neural network testing. This left me with two testable files: one with 10,000 features (points) and one with 100 features to compare accuracy, network generation time, and input nodes.

5.2 Neural Network Results

I tested my 822 collected hand drawn alphabet samples on Weka Explorer to evaluate the success of various neural networks. I generated two versions of each set of data: one with 10,000 features and one with 100 features. I tested the samples on a multi-layer perceptron network, a 5 decision tree network (J48), and a random tree network. Table 5.1 displays the results from testing the alphabet on the 10,000 feature data. Table 5.2 displays the results from testing the 100 feature alphabet on these networks.

Table 5.1: Weka testing results of 10,000 feature alphabet.

Neural Network	Accuracy	Build Time	Size	Testing Time
Mulit-Layer Perceptron	N/A	Time-out >30 min	N/A	N/A
J48 Pruned Tree	96.2287%	4.07 sec	199	0.06
Random Tree	100%	0.18 sec	629	0.43 sec

Table 5.2: Weka testing results of 100 feature alphabet.

Neural Network	Accuracy	Build Time	Size	Testing Time
Mulit-Layer Perceptron	99.8783%	102.96 sec	130 nodes	0.28 sec
J48 Pruned Tree	97.3236%	0.22 sec	161	0.04
Random Tree	100%	0.04 sec	387	0.03

The number of selected attributes was unable to be determined for the 10,000 feature alphabet due to a time-out by Weka after 30 minutes. The 10-by-10 block alphabet has 65 select attributes that are unique and determine each letter.

5.3 Geometric Shape Recognition

The geometric shape recognizer displayed relative success in identifying hexagons and lines.

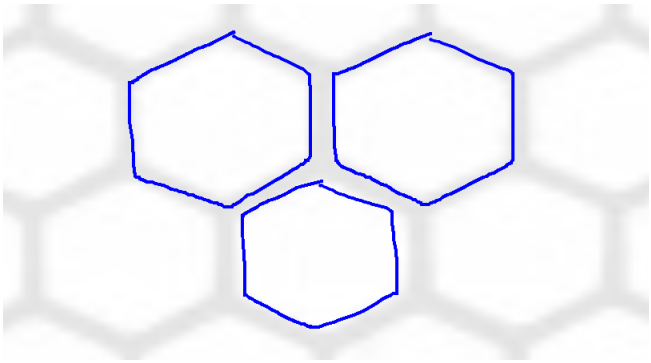

Figure 5.2: Correctly identified hexagons.

Figure 5.2 shows three examples of correctly identified hexagons. These hexagons vary in size and sharpness of corners but the recognizer was able to identify them as hexagons. Some hexagons that were not recognized were drawn with very curved corners or had a lot of noise throughout the stroke.

Lines were recognized with about 75% accuracy. The line identifier relied on evaluating how many points were on the slope of the theoretical line drawn between the first and last point of the stroke. Lines that had a hook on the end or that were mostly vertical were a challenge to identify.

Figure 5.3 shows three examples of false negative lines by the system. These lines

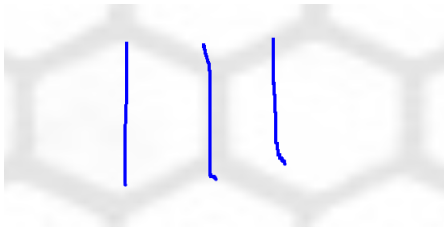

Figure 5.3: False negative lines.

should obviously be identified as such, but due to the vertical nature, they have been misidentified. The second and third line of the image have slight hooks at the bottom. This is a significant factor as to the false negative of the system. Perfect vertical lines that were drawn straight and without hooks were correctly identified by the system.

6. DISCUSSION

The focus of this chapter is to expand on the results from the prior chapter with my insights that interpret the results, as well as the successes and challenges from the thesis work.

6.1 Insights

6.1.1 Weka Testing Comparisons

The Weka testing of the neural networks yielded many interesting results. The testing of the 10,000 feature network only failed for the multi-layer perceptron test. The system timed-out after 30 minutes of trying to build a neural network corresponding to the supplied data. This is because each of the 822 samples had 10,000 individual features of 1's and 0's and building a successful neural network on all this data would require many hidden layers of nodes. The system simply could not generate a neural network that would have relative accuracy during the testing phase due to the inundation of data. However, when testing the 10,000 feature data on other neural networks, the system was able to generate neural networks that had strong accuracy. The J48 pruned tree network only took 4.07 seconds to build the 199 size tree with 100 leaves. The accuracy of this network was 96.2287%, which suggests that Weka chose building a less accurate tree was more significant than timing out on the input data set. By comparison, the J48 pruned tree of the 100 feature alphabet had 97.3236%. This was only slightly above the accuracy of the 10,000 feature alphabet, but the time to build the 100 feature network was over 18 times faster than building a network for the 10,000 feature network. The random tree network for the 10,000 feature and 100 feature alphabet both had an accuracy of 100%, but the differences lie in the build time and size. The 100 feature network was over four times

faster and the size of the tree was about 60% the size of the 10,000 feature network. When considering the memory allocation needed to store the network, the 100 feature network would be ideal since it is considerably smaller than the 10,000 feature network with the same amount of accuracy.

When comparing the results of the 10,000 feature network versus the 100 feature network, the 100 feature network is preferred for this application. The network build time for all the tested networks for the 100 feature data were significantly faster than the 10,000 feature data. Also, the data allocation for the 100 feature network was smaller for each of the network tests. This would optimize the system by decreasing necessary memory allocation for the network. Also, the testing time for each of the 100 feature networks was less than that of the 10,000 feature networks, indicating that a 100 feature network would yield faster results than a 10,000 feature network.

6.1.2 100 Feature Multi-Layer Perceptron Network

The multi-layer perceptron network for the 100 feature network did not time-out like the 10,000 feature network did and had 99.8783% accuracy with a 102.96 second build time. Evaluating the results closely revealed that the one incorrectly identified instance was misinterpreting a capital 'O' as a lowercase 'o'. Since both these instances are interpreted the same on a 100-by-100 pixel grid, the network will not have to be responsible for identifying this case. Rather, an algorithm can be developed to determine relative size of these two letters to denote which is intended as a lowercase 'o' versus a capital 'O'. Therefore, it might be better to train two separate neural networks - one for capital letters and one for lowercase letters. The recognizer will be responsible for identifying when a collection of strokes is considered a capital letter versus a lowercase letter and send the collection of strokes to the respective neural network for identification. This will prevent similar capital and lowercase letters from being misinterpreted. For example, lowercase

letters ’c’, ’o’, ’p’, ’s’, and ’u’ are very similar to their capital letter counterparts when evaluated as a 100-by-100 pixel interpretation. Each of these letters have to be determined by their relative size when compared to other determined letters to determine if they are capital or lowercase letters.

6.2 What Worked

6.2.1 Geometric Recognizer

The geometric shape recognizer displayed relative success during testing. The conditions of identification of a hexagon were able to correctly identify when a stroke was a hexagon. Lines were also identified with relative success. However, when strokes were drawn very quickly and with significant noise, the recognizer was unable to successfully identify the shape. Using the condition that overlapping bounding boxes indicated strokes should be grouped together worked to identify when strokes were intersecting. The algorithm for determining overlapping strokes worked with near-perfect accuracy. Since multiple stroke alphanumerics were assumed to be drawn with intersecting strokes, the algorithm for overlapping bounding boxes was significant for collecting data to be sent to the neural network for identification.

6.2.2 Alphanumeric Recognizer

The neural network for recognizing alphanumerics worked with significant success. The results were 95% and above for each of the networks tested, with two of the networks reaching 100%. The 100 feature neural network had the best times overall (build time and test time), which is preferred for optimization of the system.

6.3 Challenges

6.3.1 Neural Networks and GUI

The 10,000 feature neural network did not work. The large testing data overloaded the system and failed to generate a neural network before the system timed out. A challenge of working with the GUI system is that the top left point is the origin (0,0) whereas common mathematical orientation has the bottom left point being the origin. This caused trouble until I was able to flip the orientation in my head where the x plane increased horizontally right and the y plane increased vertically down. Once I had adjusted my orientation of width and height, it was significantly easier to code algorithms to generate bounding boxes and collect matrix data for testing the neural networks.

6.3.2 Vertical Line Recognizer

My original line recognizer did not work with sufficient success. Vertical lines continually proved false positive since the slopes of these lines were positive or negative infinity. This meant that if the stroke varied slightly through a curve between the first and last point, the recognizer was unable to find enough points on the ideal theoretical line to deem the stroke a line. Similarly, if the line contained a hook at the end, the recognizer would fail to identify the stroke as a line. This is because the hook would alter the ideal theoretical line, and not enough points of the stroke fell on this theoretical line. Therefore, a better algorithm would take the slopes from neighboring points rather than the slope of the entire line for comparison. Also, the line recognizer needs to account for the possibility of a hook on the line and correctly remove it from the line before attempting to identify it. The line recognizer needs to be capable of removing noise such as a hook or accidental curve in order to reduce the amount of false positives.

7. FUTURE WORK

7.1 SMILES Conversion and Correctness

A main component of this research was correctness evaluation of a chemical structure. However, due to time restraints and a shifted focus on the sketch recognition portion, the current software does not allow for structure correctness evaluation. A correct SMILES conversion would generate a line of characters and symbols to represent the chemical structure representation [47, 48]. The PubChem database has over 100 million chemical molecules and details about each of these compounds, including each molecule's SMILES notation [49]. Future research will be directed towards developing a personal algorithm to translate a chemical structure to its SMILES notation. The SMILES tutorial provides several insights into this conversion technique [50].

There are also several online converters that convert a chemical structure to its respective SMILES notation. Once the sketch is parsed as its individual components, an ideal version of that component can be generated and placed in its respective location on a blank canvas. The canvas can then be saved as a png or jpeg and input the generated photo through an online SMILES conversion software. The output SMILES notation will then be compared against the structure's expected SMILES to analyze correctness.

Another main goal of the SMILES conversion and comparison is to output where mistakes lie in the structure. A simple analysis of the mistakes would output as text what is wrong. For example, if the structure is missing an element, output what element is missing. If the structure has an element in the wrong place, output which element is incorrectly placed in the current structure. A better version of this error analysis would graphically display which element is incorrect.

Figure 7.1 displays an example of a graphical error message that clearly displays where

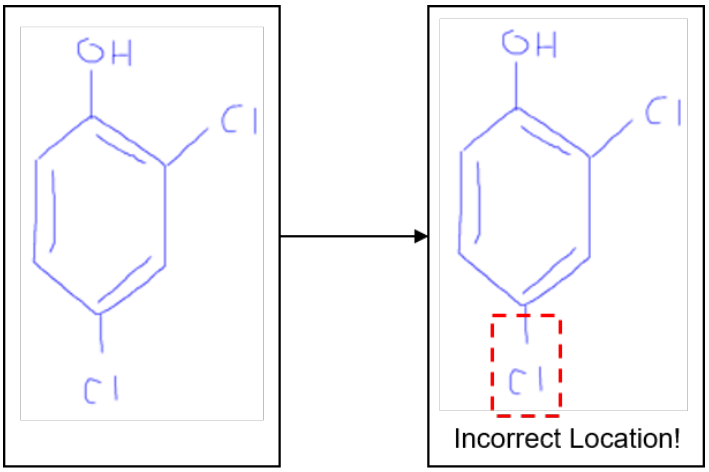

Figure 7.1: Ideal error detection.

the error is located and what the error is that is causing the structure to be incorrect. This would be the ideal method for error analysis since this method is not available on the current drag-and-drop ITS. Current ITS only state what the error is, not where it is located in the user provided chemical structure. This would add an especially helpful component for students struggling to learn how to draw correct chemical structures.

7.2 Stereochemistry

Future work can be done to increase the realm of possible chemical structures drawn by including structures with stereochemistry. SMILES encompass stereochemistry, so the real challenge would lie in the sketch recognition of the hash and wedge bonds that are unique to stereochemistry. Both hash and wedge bonds require multiple strokes that may be misinterpreted as letters or lines representing shared electrons. A new algorithm would have to be developed to recognize these bonds without interfering with the recognition algorithms for other structural components. The hash bonds in particular would be extremely difficult for a sketch recognition algorithm to interpret. The hash bonds are a series of lines, increasing in size as the bond extends to a certain chemical element or

compound.

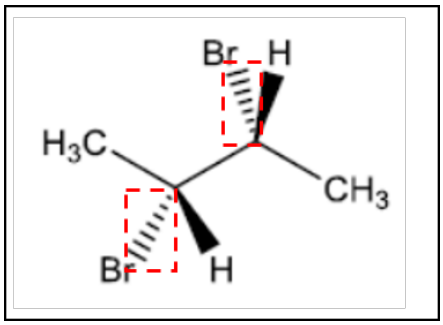

Figure 7.2: Challenges of stereochemistry hash bonds.

Figure 7.2 shows examples of these hash bonds. Each stroke of the hash bond would have no definable connection to the previous line of the hash bond. This would make it particularly difficult for an algorithm to grab all the strokes of the hash bond for correct identification. There is also no predefined number for how many strokes comprise a hash bond. A hash bond typically has three strokes but can also have many more as Figure 7.2 displays. If a single line is not included, it could be misinterpreted later as a shared electron bond or the letter 'l'.

Another challenge to adding stereochemistry would be altering the algorithm that converts chemical structures into their corresponding SMILES form. The SMILES enumeration can account for stereochemistry, but chemical elements of a stereochemical configuration can be flipped any number of ways, as long as the hash and wedge bonds are also flipped accordingly. A new algorithm would have to be generated that could create SMILES forms for each possible versions of the stereochemical structure, then compare to see if one of the variations matches the SMILES on file for that stereochemical compound.

7.3 Image Recognition

The current software involves a user drawing directly on a touch screen. Touch screens such as phones and computers are widely available today, but not every student owns one. Therefore, future work could be done to extend the software to uploaded images. This would require an image recognition algorithm that is capable of interpreting the written component of an image and translate this to a grid. Once the image is correctly input as points on a grid, the software would be able to run to identify the components of the structure and the correctness of the structure. With image recognition of hand drawn sketches, there are several challenges. One challenge is identifying strokes versus erroneous lines that were erased. The erased lines can leave faded marks which might be misinterpreted by the program. Also, if the uploaded image is blurry, the image recognition may misinterpret components of the unclear structure.

8. CONCLUSION

A subject such as Organic Chemistry requires sufficient practice to learn the comprehensive rules of the domain. Sketch-based intelligent tutoring systems can ease student understanding and direct useful practice in various topics of Organic Chemistry. User feedback is pivotal for student's understanding and improvement. Neural networks can be integrated with sketch recognition to allow for a wide range of hand drawn chemical components. The sketch recognizer parses the chemical structure to determine if the stroke or combination of strokes are geometric shapes or alphanumerics. Geometric shapes are identified by specific algorithms and alphanumerics are identified by neural networks. The correctness of the user drawn structure can be calculated by comparison with structures from a large chemistry database.

REFERENCES

[1] A. O'Dwyer and P. Childs, "Organic chemistry in action! developing an intervention program for introductory organic chemistry to improve learners' understanding, interest, and attitudes," *J Chem Educ*, vol. 91, no. 7, pp. 987–993, 2014.

[2] T. N. Hrin, D. D. Milenković, and M. D. Segedinac, "The effect of systemic synthesis questions [ssynqs] on students' performance and meaningful learning in secondary organic chemistry teaching," *Int J of Sci and Math Educ*, vol. 14, pp. 805–824, 2016.

[3] J. Ealy, "Analysis of students' missed organic chemistry quiz questions that stress the importance of prior general chemistry knowledge," *Educ Sci*, vol. 8, no. 42, pp. 1–13, 2018.

[4] B. Bruner, *ChemSketch - An Introductory Guide*, 2014.

[5] T. Y. Ouyang and R. Davis, "Recognition of hand drawn chemical diagrams," in *Proceedings of AAAI 2007*, pp. 846–851, 2007.

[6] E. V. Anslyn and D. A. Dougherty, *Modern Physical Organic Chemistry*. Herndon, VA, USA: University Science Books, 2005.

[7] C. Voss, *Modeling Molecules with Recurrent Neural Networks*, 2017. http://csvoss.github.io/projects/2015/10/08/rnns-and-chemistry.html.

[8] J. Gasteiger and J. Zupan, "Neural networks in chemistry," in *Angewandte Chemie International Edition in English*, vol. 32, pp. 503–527, 1993.

[9] J. Wei, D. Duvenaud, and A. Aspuru-Guzik, "Neural networks for the prediction of organic chemistry reactions," *American Chemical Society Publications*, 2016.

[10] A. D. Kulkarni, *Artificial Neural Networks for Image Understanding*. New York, NY, USA: John Wiley & Sons, Inc., first ed., 1993.

[11] M. Blumenstein and B. Verma, *A Neural Network for Real-World Postal Address Recognition*. London, UK: Springer, 1998.

[12] R. Al-Jawfi, "Handwriting arabic character recognition lenet using neural network," *Int. Arab J. Inf. Technol.*, vol. 6, pp. 304–309, 2009.

[13] T. Hammond, "Enabling instructors to develop sketch recognition applications for the classroom," in *Frontiers in Education Conference*, vol. 4, (Milwaukee), ASEE/IEEE Frontiers in Education Conference, October 2007.

[14] T. A. Hammond, B. Eoff, B. Paulson, A. Wolin, K. Dahmen, J. Johnston, and P. Rajan, "Free-sketch recognition: putting the chi in sketching," in *CHI Extended Abstracts*, 2008.

[15] J. Peschel and T. Hammond, "Strat: a sketched-truss recognition and analysis tool," in *International Workshop on Visual Languages and Computing*, pp. 282–287, January 2008.

[16] S. Valentine, F. Vides, G. Lucchese, D. Turner, H. hoe Kim, W. Li, J. Linsey, and T. Hammond, "Mechanix: A sketch-based tutoring and grading system for free-body diagrams," *AI Magazine*, Winter 2012.

[17] T. Nelligan, S. Polsley, J. Ray, M. Helms, J. Linsey, and T. Hammond, "Mechanix: A sketch-based educational interface," in *Proceedings of the 20th International Conference on Intelligent User Interfaces Companion*, IUI Companion '15, (New York, NY, USA), pp. 53–56, ACM, 2015.

[18] E. C. Hilton, B. WilliFord, W. Li, E. McTigue, T. Hammond, and J. Linsey, "Consistently evaluating sketching ability in engineering curriculum," in *4th ICDC: International Conference on Design and Creativity*, November 2016.

[19] B. Paulson, B. Eoff, A. Wolin, J. Johnston, and T. Hammond, "Sketch-based educational games: "drawing" kids away from traditional interfaces," in *Proceedings of the 7th International Conference on Interaction Design and Children*, IDC '08, (New York, NY, USA), pp. 133–136, ACM, 2008.

[20] F. Vides, P. Taele, H.-H. Kim, J. Ho, and T. Hammond, "Intelligent feedback for kids using sketch recognition," in *Computer Human Interaction Works-In-Progress*, 2012.

[21] H. hoe Kim, P. Taele, S. Valentine, E. McTigue, and T. A. Hammond, "Kimchi: A sketch-based developmental skill classifier to enhance pen-driven educational interfaces for children," in *Sketch Based Interfaces and Modeling*, 2013.

[22] H. Kim, P. Taele, S. Valentine, and T. Hammond, "Developing intelligent sketch-based applications for children's fine motor sketching skill development," in *Proceedings of the 2014 International Conference on Intelligent User Interfaces Workshop on Sketch: Pen and Touch Recognition*, IUI SKETCH '14, February 2014.

[23] H. Kim, P. Taele, J. H. Seo, J. Liew, and T. Hammond, "A novel sketch-based interface for improving children's fine motor skills and school readiness," in *Expressive '16: Proceedings of the Joint Symposium on Computational Aesthetics and Sketch Based Interfaces and Modeling and Non-Photorealistic Animation and Rendering*, Expressive '16, pp. 69–78, Eurographics Association, May 2016.

[24] P. Taele and T. Hammond, "Hashigo: A next-generation sketch interactive system for japanese kanji," in *IAAI*, 2009.

[25] N. Shahzad, B. Paulson, and T. Hammond, "Urdu qaeda: Recognition system for isolated urdu characters," in *Proceedings of the Workshop on Sketch Recognition at the 14th International Conference of Intelligent User Interfaces*, IUI SR '14, pp. 1–5, 2009.

[26] P. Taele and T. Hammond, "Lamps: A sketch recognition-based teaching tool for mandarin phonetic symbols i," *Journal of Visual Languages & Computing*, vol. 21, pp. 109–120, April 2010.

[27] P. Taele and T. A. Hammond, "Boponoto: An intelligent sketch education application for learning zhuyin phonetic script," in *The 21st International Conference on Distributed Multimedia Systems*, 2015.

[28] P. Taele and T. Hammond, "Enhancing instruction of written east asian languages with sketch recognition-based "intelligent language workbook" interfaces," in *The Impact of Pen and Touch Technology on Education*, pp. 119–126, Springer, July 2015.

[29] D. Sloniger, R. Sakariya, C. D. Guzman, J. Mathews, E. Susanto, J. I. Koh, P. Taele, and T. Hammond, "Tensai: A sketch-based educational game for assessing proper writing of japanese writing scripts," in *44th International Conference on Graphics, Visualization & Human-Computer Interaction*, GI '18, (Waterloo, Ontario, Canada), Canadian Human-Computer Communications Society, 2018.

[30] T. Chu, P. Taele, and T. Hammond, "Supporting chinese character educational interfaces with richer assessment feedback through sketch recognition," in *44th International Conference on Graphics, Visualization & Human-Computer Interaction*, GI '18, (Waterloo, Ontario, Canada), Canadian Human-Computer Communications Society, 2018.

[31] W. Li, E. Hilton, T. Hammond, and J. Linsey, "Persketchtivity: An intelligent pen-based online education platform for sketching instruction," in *Proceedings of EVA London 2016: Electronic Visualisation & the Arts*, pp. 133–141, July 2016.

[32] S. Keshavabhotla, B. Williford, S. Kumar, E. Hilton, P. Taele, W. Li, J. Linsey, and T. Hammond, "Conquering the cube: learning to sketch primitives in perspective with an intelligent tutoring system," in *Proceedings of the Symposium on Sketch-Based Interfaces and Modeling*, pp. 1–11, July 2017.

[33] B. Williford, M. Runyon, A. H. Malla, W. Li, J. Linsey, and T. A. Hammond, "Zensketch: A sketch-based game for improving line work," in *CHI PLAY*, 2017.

[34] B. Williford, "Sketchtivity: Improving creativity by learning sketching with an intelligent tutoring system," in *Creativity & Cognition*, 2017.

[35] P. Taele, L. Barreto, and T. Hammond, "Maestoso: An intelligent educational sketching tool for learning music theory," in *AAAI*, 2015.

[36] L. Barreto, P. Taele, and T. Hammond, "A stylus-driven intelligent tutoring system for music education instruction," in *Revolutionizing Education with Digital Ink* (T. Hammond, S. Valentine, and A. Adler, eds.), pp. 141–161, Springer, May 2016.

[37] J. J. LaViola, Jr. and R. C. Zeleznik, "Mathpad2: A system for the creation and exploration of mathematical sketches," in *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pp. 432–440, ACM, August 2004.

[38] J. J. LaViola, "An initial evaluation of a pen-based tool for creating dynamic mathematical illustrations," in *Proceedings of the Third Eurographics Conference on Sketch-Based Interfaces and Modeling*, SBM'06, (Aire-la-Ville, Switzerland, Switzerland), pp. 157–164, Eurographics Association, 2006.

[39] J. J. LaViola, "Advances in mathematical sketching: Moving toward the paradigm's full potential," *IEEE Computer Graphics and Applications*, vol. 27, pp. 38–48, January 2007.

[40] D. Dixon, M. Prasad, and T. Hammond, "icandraw: Using sketch recognition and corrective feedback to assist a user in drawing human faces," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pp. 897–906, ACM, April 2010.

[41] D. Cummmings, F. Vides, and T. Hammond, "I don't believe my eyes!: Geometric sketch recognition for a computer art tutorial," in *Proceedings of the International Symposium on Sketch-Based Interfaces and Modeling*, SBIM '12, pp. 97–106, Eurographics Association, June 2012.

[42] T. Hammond, *LADDER: A Perceptually-Based Language to Simplify Sketch Recognition User Interfaces Development*. PhD thesis, Massachusetts Institute of Technology, 2007.

[43] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *International 1989 Joint Conference on Neural Networks*, vol. 1, (Washington, D.C., USA), pp. 593 – 605, 1989.

[44] J. O. Wobbrock, A. D. Wilson, and Y. Li, "Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes," in *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, UIST '07, pp. 159–168, ACM, October 2007.

[45] B. Paulson and T. Hammond, "Paleosketch: Accurate primitive sketch recognition and beautification," in *Proceedings of the 13th International Conference on Intelligent User Interfaces*, IUI '08, pp. 1–10, ACM, January 2008.

[46] A. Wolin, B. Eoff, and T. Hammond, "Shortstraw: A simple and effective corner finder for polylines," in *SBM*, 2008.

[47] D. C. I. Systems, *SMILES - A Simplified Chemical Language*, September 2007.

[48] E. J. Bjerrum, "Smiles enumeration as data augmentation for neural network modeling of molecules." Wildcard Pharmaceutical Consulting.

[49] M. Swain, *PubChemPy 1.0*, 2013. https://pypi.python.org/pypi/PubChemPy/1.0.

[50] D. C. I. Systems, *SMILES Tutorial*, September 2007.

[51] M. Caudill, "Neural network primer: Part 1." AI Expert, February 1989.

[52] J. M. Zurada, *Introduction to Artificial Neural Networks*. West Publishing Company, 1992.

[53] W. A. M. Ahmed, E. S. M. Saad, and E. S. A. Aziz, "Modified back propagation algorithm for learning artificial neural networks," in *Radio Science Conference, 2001*, vol. 1, (Mansoura, Egypt), pp. 345 – 352, 2001.

[54] S. Feteih and D. Sadhukkan, "Training strategy for back-propagation neural networks using input weighting," in *American Control Conference*, vol. 2, pp. 1384 – 1385, 1994.

APPENDIX: NEURAL NETWORKS

8.1 Introduction

Modern computing infuses biological interactions to create realistic neurocomputing devices capable of mimicking natural behaviors. In the past two decades, artificial neural networks have been developed that are capable of understanding human interaction based on advanced learning algorithms. Defined by one of the first neurocomputer inventors Robert Hecht-Nielsen, an artificial neural network is "a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs" [51]. Back-propagation neural networks are ANNs with the basis of a back-propagation that calculates the error between the input and expected output in order to optimize the accuracy in which it anticipates the correct output. The addition of the back-propagation algorithm in ANNs has allowed for back-propagation neural networks to be implemented in modern learning simulations. For example, back-propagation neural networks have been implemented to recognize human handwriting with impeccable accuracy [52]. Back-propagation neural networks have significantly advanced the current state of artificial intelligence and can be further expanded to other fields.

8.2 Back-Propagation Neural Network Structure

The basis of back-propagation neural networks lies in the approximation function uses to map inputs to their respective outputs. In simple implementations of a back-propagation neural network, one layer of input nodes is mapped to the layer of potential outputs.

Figure 8.1 displays a simple diagram denoting the input neurons mapped to the respective output neurons. Each individual neuron is applied through a mapping function,

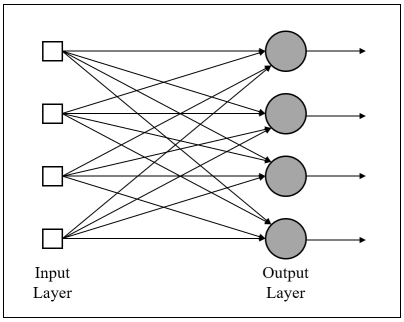

Figure 8.1: Single layer neural network.

weighted based on the input itself, applied against the sigmoid function, then mapped to the output corresponding to the final value. The most basic back-propagation neural networks involve a single layer of input neurons mapped to the output layer. In more complex implementations of a back-propagation neural network, hidden layers are implemented between the input and output neuron layers, allowing for higher accuracy to be achieved.

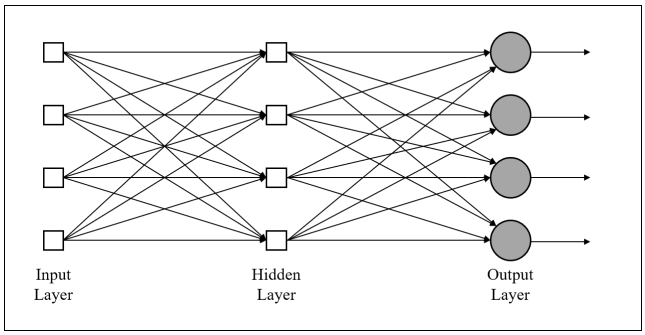

Figure 8.2: Multi-layer neural network

Figure 8.2 demonstrates a multi-layer neural network design. There is an initial input layer where each input neuron is connected to each neuron of the hidden layer. Each neuron of the hidden layer is then connected to each neuron of the output layer. This

webbed design between the input layer and hidden layer allows for much higher accuracy since the weight algorithm of each neuron is now applied for each hidden layer. Thus, more hidden layers will result in higher accuracy [53].

8.3 Back-Propagation Neural Network Learning

In a feedforward neural network, the output of a layer of neurons is used at the input to the subsequent layer of neurons. This results in a cascading connected network to create a multi-layer network. Hidden layers of neurons in these feedforward networks increase accuracy by providing more weighted connections to activate the output neurons [43, 54]. Feedforward networks are often described as static since they only process signals in a one-way direction. As the network is being trained on a set of data, each layer of neurons is filled with the input data. The network updates the weights of each neuron using two functions - one to calculate the signal error for all nodes in the hidden layers and output layer and one to calculate the error of the weights and bias. The neural network iterates through each layer to determine the signal error for each node by comparing the node's current output with its expected output. The network works backwards to calculate the signal error of all the nodes in the hidden layers. The sum of each layer's signal error is calculated then the complement of this sum is multiplied by each node of that layer to calculate that node's corresponding signal error. The network calculates the bias weight of each node as that node's signal error multiplied by the learning rate and added to the momentum multiplied by the node's threshold difference. This bias weight determines the node's significance in identifying an output. The overall error is a significant factor for determining if training is complete. The network compares the expected output with the current output of the neural network to determine the error of the entire system. The network training is deemed complete when this error is below a predetermined threshold.